UDC 004.41.2

Adapting the Unified Software Development Process for User Interface Development

Željko Obrenović[1], Dušan Starčević[1]

[1] Laboratory for Multimedia Communications,
School of Business Administration,
University of Belgrade, Belgrade, Serbia and Montenegro
{obren, starcev}@fon.bg.ac.yu

Abstract. In this paper we describe how existing software developing processes, such as Rational Unified Process, can be adapted in order to allow disciplined and more efficient development of user interfaces. The main objective of this paper is to demonstrate that standard modeling environments, based on the UML, can be adapted and efficiently used for user interfaces development. We have integrated the HCI knowledge into developing processes by semantically enriching the models created in each of the process activities of the process. By using UML, we can make easier use of HCI knowledge for ordinary software engineers who, usually, are not familiar with results of HCI researches, so these results can have broader and more practical effects. By providing a standard means for representing human-computer interaction, we can seamlessly transfer UML models of multimodal interfaces between design and specialized analysis tools. Standardization provides a significant driving force for further progress because it codifies best practices, enables and encourages reuse, and facilitates interworking between complementary tools. Proposed solutions can be valuable for software developers, who can improve quality of user interfaces and their communication with user interface designers, as well as for human computer interaction researchers, who can use standard methods to include their results into software developing processes.

1. Introduction

Interaction and usability aspects of software systems are becoming more relevant and are often identified as one of the critical software quality attributes. Therefore, software developers have started to pay more attention to design of user interfaces, trying to exploit HCI techniques aimed to produce a useful software product. However, integration of HCI knowledge into the software development processes is not straightforward, and often leads to solutions that are not widely used. At this point, main challenge is to make

easier use of HCI knowledge for ordinary software engineers who, usually, are not familiar with many results of HCI researches.

In this paper we describe how existing software developing processes, such as Rational Unified Process, can be adapted in order to allow disciplined and more efficient development of user interfaces. The main objective of this paper is to demonstrate that standard modeling environments, based on the UML, can be adapted and efficiently used for user interfaces development. We have integrated the HCI knowledge into developing processes by semantically enriching the models created in each of the software development process activity. This paper is primarily aimed for software developers, especially for those who develop complex multimodal user interfaces. Therefore, we wanted to provide a more illustrative and practical solution, so that developers can more easily apply it in their work. Using industry standards, such as the UML and Unified process, which are widespread accepted throughout the software community, provides a good opportunity to advance developing of user interfaces. Incorporating a user interface extensions into the UML gives us a standard way of producing formal models of human-computer interaction. Using wide accepted standards provides a significant impetus for further progress because it codifies best practices, enables and encourages reuse, and facilitates interworking between complementary tools. By providing standard means for representing human-computer interaction, we can seamlessly transfer UML models of interfaces between design and specialized analysis tools [1].

In next section, we briefly describe some of the existing solutions. After that, we outline the basic idea of our approach. Then, one by one, we present proposed semantic extensions of software development process activities, where we describe requirements specification, analysis, design, implementation and testing. In the end, we give short discussion and conclusions.

2. Existing solutions

Software development and user interface researches have a long and intertwined history. User interface community has developed and used various software tools, such as window managers and toolkits, interactive graphical tools, component systems, event languages, scripting languages, hypertext, object-oriented programming, but also some promising but less successful products, such as, user interface management systems and formal language-based tools. However, these existing user interface software tools solve just a part of the human-computer interaction problem. In addition, they use specialized notation and conventions, what limits their practical usage

and complicates integration of the user interface with the rest of the software system. Furthermore, many of these existing tools will not be able to support development of user interfaces in the future, with an increasing diversity of user interfaces on an increasing diversity of computerized devices. Existing user interfaces are mostly based on a human-computer interaction paradigm that has not changed fundamentally for nearly two decades. As the computer power has been improving exponentially, human-computer interface has become a bottleneck for many applications. This will require significant support and better and more practical integration of the HCI knowledge and software tools [2].

On the other hand, user interfaces, as primarily software artifact, may be developed using existing software developing processes. However, these processes are not adapted for particularities of user interface development and usability analysis. Consequently, there have been many attempts to integration of HCI and software development processes [3, 4, 5]. Many of these approaches proposed frameworks for improving the communication between software engineers and user interface and usability developers, enhancing object-oriented software engineering notations and models, or adapting software engineering artifacts with extensions such as annotating use case using task description. However, none of the proposed extensions of software development processes has been widely used. Also, most of those extension concentrate on extending user requirements gathering, weakly covering other activities in a software development process.

3. Adapting a Software Developing Process for User Interface Development

A software development process defines transformation of user's requirements into a software system. It defines activities and phases through which this transformation goes. According to the conceptual base they use, we can classify these processes in two main groups: structured-oriented and object-oriented [6]. Due to standardization, wide acceptance and availability of tools, we have primarily analyzed how to extend and adapt Rational Unified Process and similar object-oriented processes [7]. We will illustrate the proposed solution with the Unified process, but each of the proposed extensions can be used independently or as a part of some other software developing process, such as a developing process for small projects [8]. Developing some software system goes through many activities, and requires development of many models on different levels of abstraction. Activities often present in software development processes include [7]: requirements specification, analysis, design, implementation, and testing. In this paper we

will, in more details, requirement specification, analysis, and design, proposing practical solutions about how these activities can be adapted in order to better support development of user interfaces.

As a modeling language, we have used Unified Modeling Language (UML) [9]. UML is a good choice for modeling user interfaces for several reasons. It is a widely adopted standard that is familiar to many software practitioners, widely taught in undergraduate courses, supported by many books, and training courses. In addition, many tools from different vendors support UML. Consequently, by using UML, we can make easier use of HCI knowledge for ordinary software engineers who, usually, are not familiar with HCI researches. In this way, results of these researches can have broader and more practical effects. UML is a general-purpose modeling language, but it includes built-in facilities that allow customizations—or profiles—for a particular domain. A profile fully conforms to the semantics of general UML but specifies additional constraints on selected general concepts to capture domain-specific forms and abstractions. The creators of the UML realized that there would always be situations in which the UML, out of the box, would not be sufficient to capture the relevant semantics of a particular domain or architecture. To address this purpose, a formal extension mechanism was defined to allow practitioners to extend the semantics of the UML. The mechanism allows us to define stereotypes, tagged values and constraints that can be applied to model elements. A *stereotype* is an adornment that allows us to define a new semantic meaning for a modeling element. *Tagged values* are key value pairs that can be associated with a modeling element that allow us to "tag" any value onto a modeling element. *Constraints* are rules that define the well-formedness of a model. They can be expressed as free-form text or with the more formal Object Constraint Language (OCL) [9]. We have developed several UML profiles which we have integrated with Unified process' models. We did not wanted to create one profile, to allow tools to use just those profiles that do they want to support. When defining proposed extensions, we wanted to simplify their integration into existing projects, so that existing models do not have to be reengineered but just extended with additional elements and relations.

Our UML extensions are based on our proposal for unified model of HCI [10], shown in Figure 1. The unified model of human computer interaction is unique, formal, and standardized description of basic concepts and relations among concepts of interest to development of user interfaces. This model includes many factors such as human sensory physiology, anatomy, perception, cognition, and social interaction.

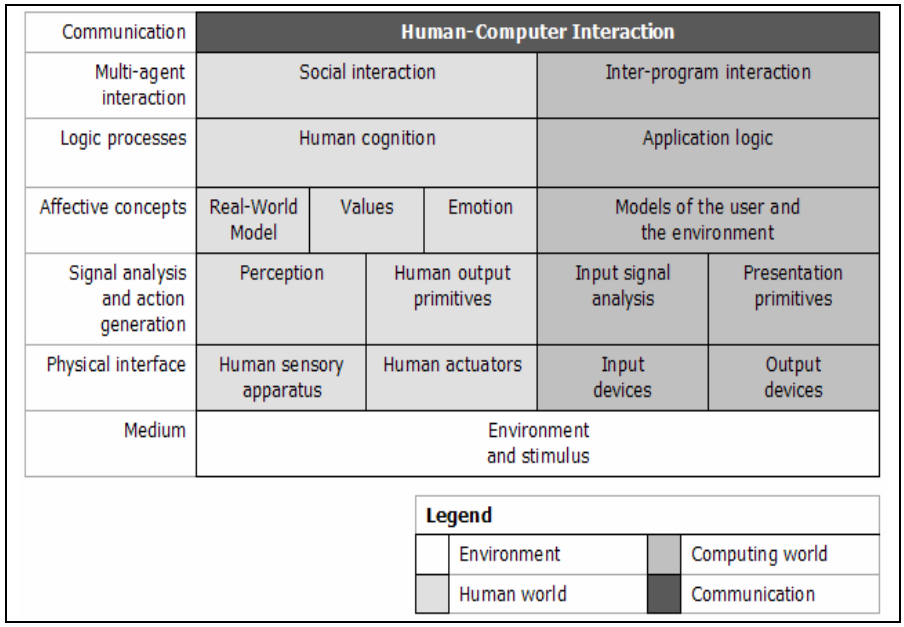


Figure 1. *Proposed unified model of HCI*.

4. Requirements specification

Requirements specification is the proces which defines what the system should do, and who will be the users of the system. Requirements specification often includes:

- *Identification of the system environment*, e.g. the context in which the system should work,
- *Recording of functional requirements*, where we achieve precise understanding of required functionality of the system,
- *Recording of nonfunctional requirements*, e.g. identification of implementation and system environment limitations, such as performance, platform dependance, and maintenance.

UML provides use-case models as principal means for requirement specification. In the Unified process, for example, during requirements specification, we create a use-case model where we identify actors and use cases. Use-case diagrams allow simple definitions of users (and other actors), use cases and additional descriptions in a form of textual comments. Uses

cases can be specified in more details using activity diagrams or object interaction diagrams.

However, during requirements specification activity, it is not possible to specify many important factors of interest to human computer interaction. Before we can design a UI, we need information about the people who will use the tool [11]:

- Who are the system users?
- What will they need to accomplish?
- What will they need from the system to accomplish this?
- How should the system supply what they need?

These factors can be very important in other development activities, primarily in analysis and design, as many early design decisions are based on these specifications. Focusing on the user early in the development process goes a long way toward improving product quality and eliminating rework [12]. Specifying details about users and contexts is important as designers should become familiar with users' psychological characteristics (for example, cognitive abilities, motivation), level of experience, domain and task characteristics, cultural background, as well as their physical attributes (for example, age, vision, hearing) [13]. Recent initiatives such as the OMG's Model Driven Architecture (MDA) are going to make the output from requirements engineering even more significant than it is today [14].

Having in mind discussed limitations of the requirements specification activity, we propose two semantically important extensions:

- *Detailed description of users (actors),* where we can describe expected age range, skills, education, working and intellectual properties of typical users.
- *Detailed description of interaction environment,* where we can describe the location and environment of interaction. This factor is important as new mobile devices are nowadays used in various conditions and locations, with different contexts and characteristics of interaction.

For detailed description of user, we have defined the UML class stereotype <<user profile>>. Each profile is defined as a class with this stereotype, and different *tagged values*. Inside each of the user profile classes, we defined various tagged values (Table 1).

Table 1. Some tagged values for user profile classes

Profile types	Tagged values	Examples
Age profile	average range	30
	age range	[20-60]
Cognitive profile	education background	high school
	primary education area	mechanics
Social profile	cultural background	western
Linguistics profile	languages	Serbian, English
Disability profile	sight	normal or corrected to normal
	sight colors	normal
	hearing	normal
	movements	normal or wheelchair support
	hand usage	normal
	data processing	all
	reading skills	average

Profiles created in this way can be connected with actor classes in various ways. We suggest using the inheritance relation, where we could define that some actors inherits one or more user profiles. In this way we can specify user groups than will use the system. We have also defined an inheritance stereotype named <<typical user>>, which denotes which of the profiles, if any, represents a typical user of the system. Figure 2a illustrates use of the proposed extensions.

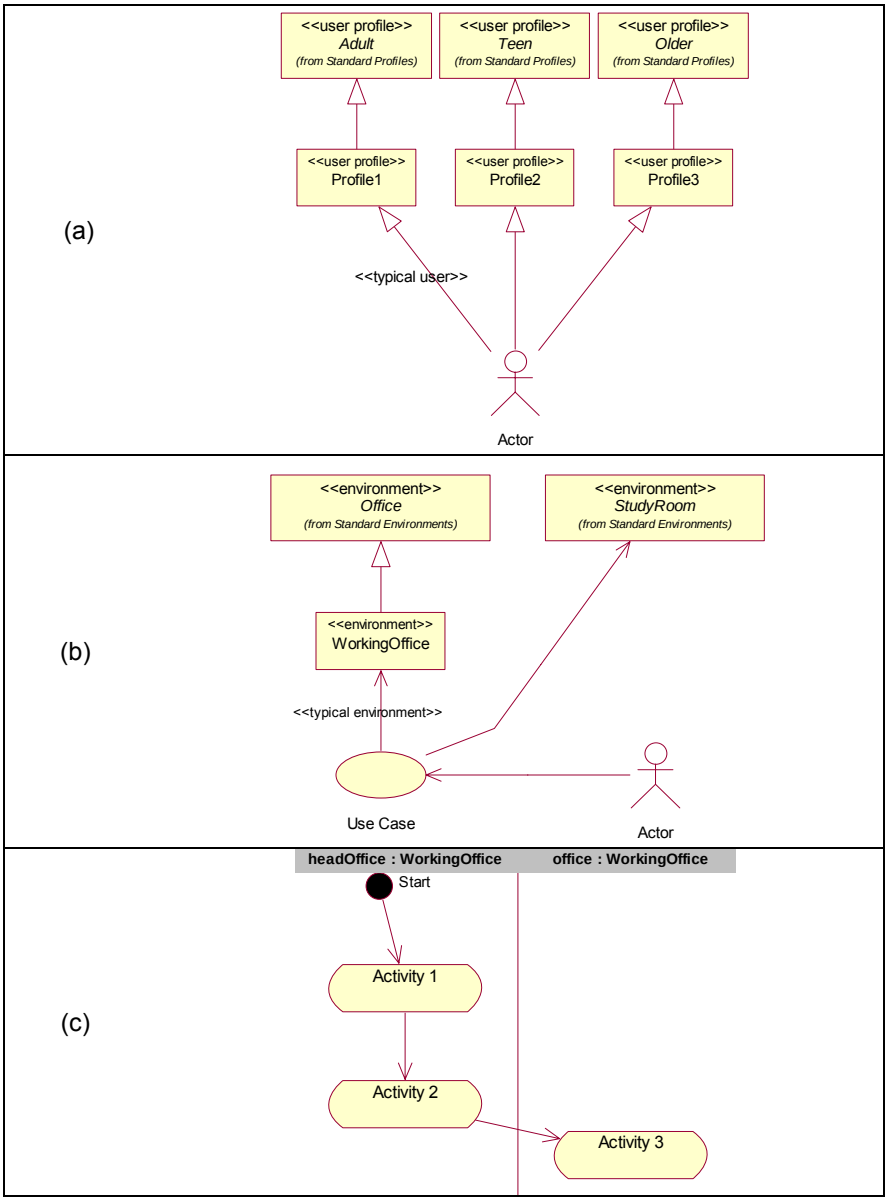


Figure 2. Examples of detailed description of an actor (a), and environment in use case diagrams (b) and in activity diagrams (c)

For detailed description of interaction environment, we have defined a class stereotype <<environment>>, as well as additional tagged values for

description of environment conditions such as visibility, average noise level, average temperature, air pressure, and humidity. We propose connecting the environment classes with use cases with association relation. A use case can be realized in many environments, and one environment can be used for more use cases. Additionally, we have defined an association stereotype <<typical environment>>, which denotes the environment class which represents typical environment for realization of some use case (Figure 2b). More detailed description of the use case realization can be achieved using activity diagrams. In these diagrams, the environment classes can be used to identify "swimlines", to illustrate which part of some use case will take place in some environment (Figure 2c).

5. Analysis

Analysis is the activity in which we structure and additionally describe gathered requirement specification. The main objective of the analysis is establishing clear and precise understanding of requirements, in order to make design easier. In analysis, we make platform independent, generic model of the internal organization of the system. The result of the analysis activity is the analysis model where we describe a system using analysis classes. The Unified process defines three types of these analysis classes: *boundary classes*, which describe users or other systems that will interact with the system under consideration, *control classes*, which describe some processing activity, and *entity classes*, which describe persistent elements of the system such as databases or files.

Although boundary classes can be used for simple description of user interfaces, with them it is not possible to describe many important elements of human computer interaction. As the analysis activity is primarily concerned with the creation of a platform independent description of the system, and having in mind HCI prospective, improvements for user interfaces would be *enabling platform independent, but semantically richer, description of human-computer interaction*. In this way it would be possible to specify types of communication modalities appropriate for some tasks, or used multimodal integration mechanisms. These extensions can also help a developer to analyse various general HCI decisions, for example, by connecting these models with ontologies and knowledge bases of human factors [15]. It is critical that a developer can make some analysis on highly abstract and incomplete models that arise early in the development cycle, because *this is when software designers make most of the fundamental design decisions* [1].

For a designer of software systems it is important to, early in the development cycle, get as much as possible data that can influence their

design. Therefore, we have semantically enriched analysis model by introducing two additional class stereotypes:

- <<interaction>>, defining interaction effect used in interaction, and
- <<user interface>>, which defines interactive components used for realization of user interfaces.

An <<interaction>> class, with attributes, defines the effects which designer wants to support during interaction. We have defined the following five types of effects (and corresponding attribute stereotypes): sensory, perceptual, affective, cognitive, and linguistics. Additional description of the effects can be provided inside the description of each attribute. A <<user interface>> class defines types of interface components that that designer plans to use. As in the case of interaction classes, we describe each component with an attribute. We also defined two attribute stereotypes: <<input>> i <<output>>. These stereotypes describe if the component implements presentation or user input capturing.

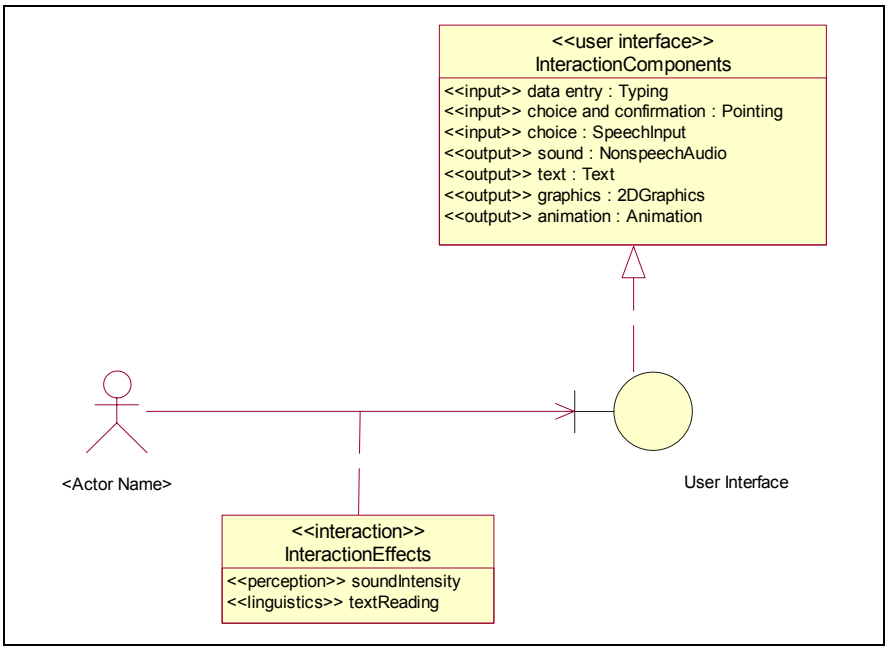


Figure 3. Usage of the proposed extensions in the analysis diagram

We propose attaching of <<interaction>> class to the relation between an actor and a boundary class that represents the user interface. On the other hand, we propose connecting that the boundary class with the <<user interface>> class using realization relation. In this way, existing analysis diagrams do not have to be reengineered but simply extended with additional elements and relations. Figure 3 illustrates how the proposed extensions can be used.

6. Design

In the design activity we shape the system in its final form, having in mind all functional and nonfunctional requirements. The main input of design activity is the result of the analysis, e.g. the analysis model. In design, we make decisions about the hardware and software implementation platform of the solution, as well as about issues such as programming languages, component systems, operating systems, distributed technologies, database technologies and user interface technologies. Also, in this activity we decompose the system into packages that are easier to control and maintain. In addition, design provides the visualization of the internal structure and functionality of the system, using chosen notation, for example, UML graphical notation. The design model contains detailed description of the structure and functionality of the system. This model is a blueprint of the solution, and it uses several UML diagrams and modeling elements, including class diagrams, object diagrams, object interaction diagrams, or state transition diagrams.

From the user interface viewpoint, the design activity requires some additional elements. User interfaces can be viewed as one-shot, higher-order messages sent from designers to users [16]. While designing a user interface the designer defines an interactive language that determines which messages and levels will be included in the interaction. Therefore, the developers need tools to describe these messages. We introduced several stereotypes for classes and relations in order to semantically enrich UML with these concepts. We introduced stereotypes such as input and output modalities, complex modes, aimed message, perceptual and cognitive effects. With these extensions, we can describe a human-computer interaction at different levels of abstraction, with various levels of details, in terms of sensory, perceptual and cognitive characteristics of various modalities. Table 2 shows some of introduced UML class and association stereotypes.

Table 2. UML design stereotypes for modeling of multimodal user interfaces.

Class stereotypes	input modality	*Defines a modality that captures some of the human outputs, such as movement or speech.*
	static output modality	*Defines a modality that statically presents data, for example, as static text, picture or graphics.*
	dynamic output modality	*Defines a modality that dynamically presents data, for example, as blinking, movie, or 3D animation.*
	human interactive response	*Defines human interactive response time scale.*
	complex modality	*Defines a modality that integrates two or more modalities.*
	visual sensory effect audio sensory effect haptic sensory effect sensory params	*Sensory effects and parameters. These effects are produced by output devices, as visual, audio or haptic stimulus.*
	human movement	*Human motor effect of movement.*
	visual perception visual 3D cue audio perception audio 3D cue haptic perception perceptual params	*Perceptual effects produced by the user interface.*
	cognitive effect analogy linguistic effect	*Cognitive and linguistic effects produced by the user interface.*
Association stereotypes	integration	*Connect a complex modality with a simple or other complex modality that it integrates.*
	effect	*Connect a modality class with sensory, perceptual or cognitive effect produced by the modality.*
	comparison	*Connects perceptual parameters with objects that are compared. Perceptual effects are always based on comparing of some basic stimulus.*
	rendering	*Connect output modalities with an output device.*
	capturing	*Connect input modality with human output that that modality captures.*

6.1. Modeling basic HCI modalities

We have made various descriptions of sensual, perceptual, motor, and cognitive effects produced by some of the widely used HCI modalities. For

example, figure 4 represents a UML class diagram, created with defined UML extensions, describing effects of graphical *textual presentation*.

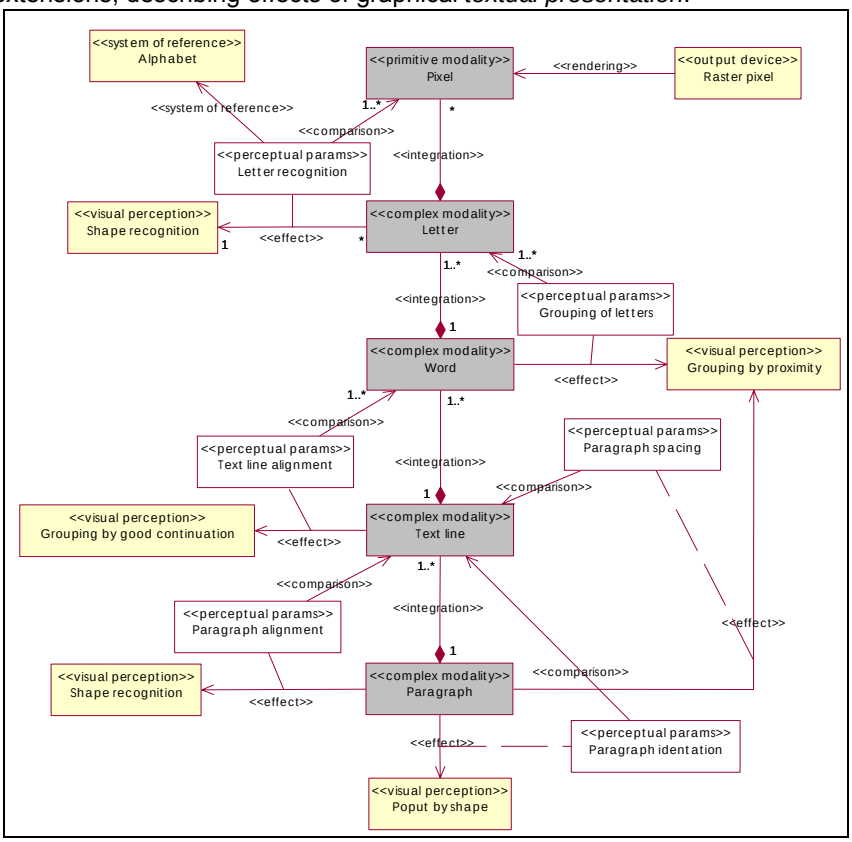


Figure 4. Description textual presentation modality

The basic modality of presentation on a screen is a pixel, rendered by some raster screen device. Pixels form letters, complex modalities that add a perceptual effect of shape recognition based on the user knowledge of the alphabet. Letters are grouped into words, which integrate letters adding perceptual effect of grouping by proximity. Words are grouped in text lines that integrate words by adding perceptual effect of grouping by good continuation. Text lines are grouped into paragraphs, that enrich presentation with several perceptual effects. Firstly, a paragraph groups text lines by proximity. Paragraph alignment changes shape of the whole paragraph. Paragraph indentation adds perceptual effect of highlighting the first line by shape, as the first line is usually shorter that other lines of text.

Another example is given in Figure 5, which describes *a table* as a presentation modality. Here, a basic presentation modality is a table cell, which introduces a visual perceptual effect of grouping by surrounding. Table cells are grouped into table lines (rows or columns), which add perceptual effects of grouping by good continuation and, optionally, grouping by surrounding (row or column borders). A table integrates lines bringing in perceptual effects of grouping by parallelism, and grouping by surrounding (table border).

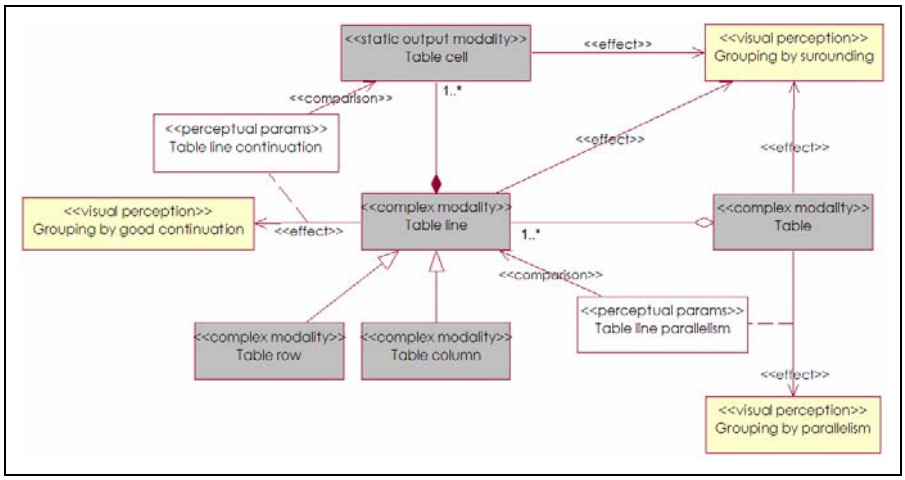


Figure 5. Description of tabular presentation modality

The last example, in Figure 6, describes *an aimed hand movement*, e.g. hand movement often used in WIMP (windows, icon, menu, pointer) interfaces. Aimed hand movement is a complex modality that integrates hand movement input, and visual feedback. Hand movement captures the movement of a user hand on a flat surface. The visual feedback is a dynamic presentation modality that animates static presentation of cursor, usually in the shape of arrow. The static cursor introduces the perceptual effect of highlighting by shape, and sometimes by depth (shadow), while a dynamic visual feedback adds perceptual effect of highlighting by motion.

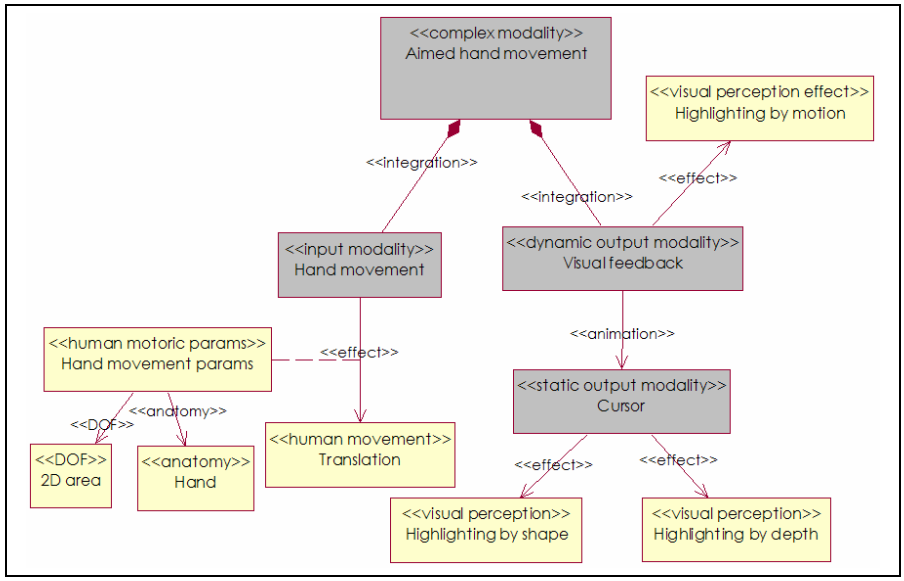
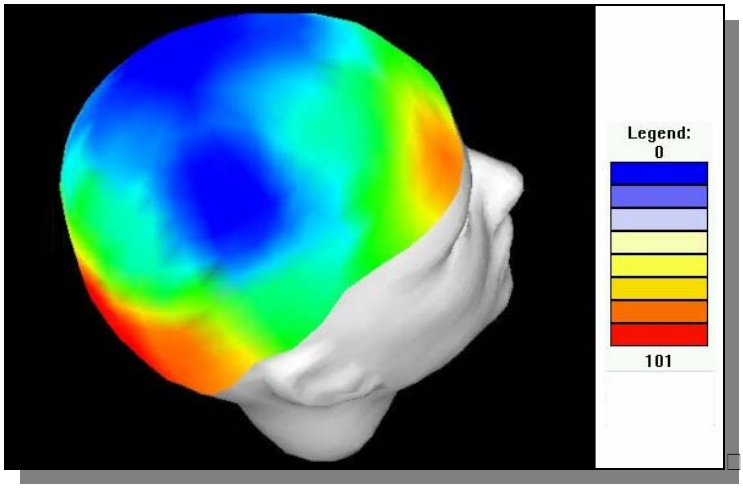


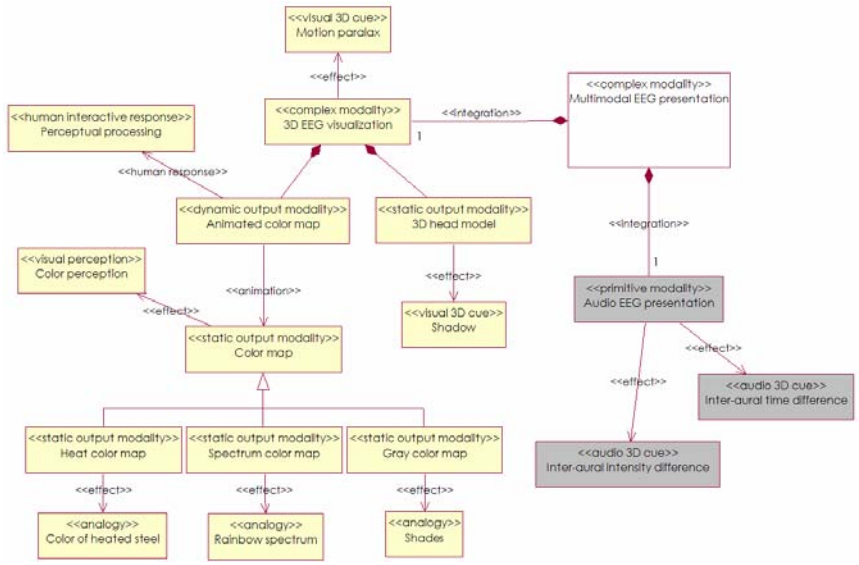
Figure 6. Description of an aimed hand movement modality

6.1 Modeling complex multimodal user interfaces

To illustrate how to create higher-level models of multimodal interfaces with our framework, we will describe the environment for multi-modal presentation of brain electrical activity. The environment, called mmViewer, utilizes various visualization and sonification modalities, facilitating efficient perceptualization of biomedical data [17]. Visualization in this environment is based on animated topographic maps projected onto the scalp of a 3D model of the head, employing several graphical modalities, including 3D presentation, animation and color (Figure 7a). Sonification is implemented as modulation of natural sound patterns to reflect certain features of processed data. Sonification emphasized the temporal dimension of the selected visualized scores. Since the visual topographic map by itself represents a large amount of visual information, sonification covered presentation of global parameters of brain electrical activity, such as the global index of left/right hemisphere symmetry. This parameter is sonified by changing position of the sound source in 3D world. Therefore, activation of a hemisphere could be perceived as movement of sound source toward that hemisphere.



(a)



(b)

Figure 7. Environment for 3D presentation of EEG signals (a), and UML class diagram of effect produced by this environment

Figure 7b presents simplified UML class diagram of perceptual and cognitive effects that the designer wanted to produce by this environment. Multimodal presentation of EEG activity is a complex modality which integrates 3D visualization and sonification. 3D visualization is by itself a complex modality which integrates a 3D head model, with an animated color map. By enabling a user to freely explore the model, 3D visualization adds motion parallax visual 3D cue. The 3D head model, by using shadow and lighting, enable users to recognize threedimensional model of the head. The animated color map animates a static color map by dynamically changing its colors based on the values of brain electric activity. This animation is smooth, fast enough to activate users' visual perceptual processing. We used three types of color maps:

- *Heat color map*, which maps values of brain electrical activity to colors from black through yellow to white, as an analogy to colors of heated steel (black means cool, red hot, white extremely hot),
- *Spectrum color map*, which uses colors of visible spectrum, from blue to red, as an analogy to rainbow spectrum familiar to many users,
- *Gray color map*, which uses different shades of gray, from black to white.
- Sonification is based on a stereo effect produced by inter-aural time and intensity difference of sound.

Using the high-level models can improve understanding of the system, but with the UML the developer can use tools for definition of transformations of these high-level and generic platform models into platform specific models which are appropriate for code generation.

8. Conclusions

In this paper we have demonstrated how extensions of the UML in various phases of the development process can enrich existing developing environments for developing of user interfaces. We have illustrated the proposed solution with the Unified process, but each of the proposed extensions can be used independently or as a part of some other software developing process, such as developing processes for small projects. We also wanted to give some "food for thought" for future research in this area.

Using industry standards, such as the UML or Unified process, which are widespread accepted throughout the software community, provides a good opportunity to advance developing of user interfaces. Incorporating a user interface extensions into the UML gives us a standard way of producing formal software models of user interfaces. Using wide accepted standards

provides a significant impetus for further progress because it codifies best practices, enables and encourages reuse, and facilitates interworking between complementary tools. By providing a standard means for representing multimodal interaction, we can seamlessly transfer UML models of multimodal interfaces between design and specialized analysis tools. Standardization provides a significant driving force for further progress because it codifies best practices, enables and encourages reuse, and facilitates interworking between complementary tools. With UML, we can jump on the bandwagon of new software development technologies, such as model driven development. Our modeling framework feats neatly in the model driven development approach, and consequently, it will be able to make use of the tools that support it. We are also working on various approaches to model transformations that could help traversing the artifacts from one design phase to another [18].

Proposed solutions can be valuable for software developers, who can improve quality of user interfaces and their communication with user interface designers, as well as for human computer interaction researchers, who can use standard methods to include their results into software developing processes.

REFERENCE

1. Bran Selic, "The Pragmatics of Model-Driven Development", IEEE Software, Vol. 20, No. 5, September / October 2003, pp. 19-25.
2. Brad Myers, Scott E. Hudson, Randy Pausch, "Past, Present, and Future of User Interface Software Tools", ACM Transactions on Computer-Human Interaction, Vol. 7, No. 1, March 2000, Pages 3-28.
3. Larry L. Constantine, "Process Agility and Software Usability: Toward Lightweight Usage-Centered Design", Constantine & Lockwood, Ltd., Report 110, 2001.
4. Natalia Juristo Juzgado, Helmut Windl, Larry L. Constantine, "Guest Editors' Introduction: Introducing Usability", IEEE Software 18(1): 20-21 (2001).
5. Xavier Ferré, Natalia Juristo, Helmut Windl, Larry Constantine, "Usability Basics for Software Developers ", IEEE Software 18(1): 22-29 (2001).
6. Roel Wieringa, "A Survey of Structured and Object-Oriented Software Specification Methods and Techniques", ACM Computer Surveys, Vol. 30, No. 4, December 1998, pp. 459-527.
7. Ivar Jacobson, Grady Booch, James Rumbaugh, The Unified Software Development Process, Addison Wesley Longman Inc., Second Printing, April 1999, ISBN 0-201-57169-2.
8. Melissa L. Russ, John D. McGregor, "A Software Development Process for Small Projects", IEEE Software, September/October 2000, pp. 96-101.

9. OMG UML™ Resource Page, http://www.uml.org/, last visited November 24, 2005.
10. Zeljko Obrenovic, "User Interfaces Development based on the Unified Model of Human Computer Interaction", PhD Thesis, Faculty of Electrical Engineering, University of Belgrade, 2004.
11. Xavier Ferré, Natalia Juristo, Helmut Windl, Larry Constantine, "Usability Basics for Software Developers ", IEEE Software 18(1): 22-29 (2001).
12. Jean Anderson, Francie Fleek, Kathi Garrity, and Fred Drake, "Integrating Usability Techniques into Software Development", IEEE Software 18(1): 46-53 (2001).
13. Leah M. Reeves, et al, "Guidelines for multimodal user interface design ", Communications of the ACM, Special issue: Multimodal interfaces that flex, adapt, and persist, Vol. 47 , No. 1 (January 2004), pp. 57 - 59.
14. Ian Graham, "The Compleat Requirements Analyste", IEEE Software, Novembar/December 2003, pp. 99-101.
15. Zeljko Obrenovic, Dusan Starcevic, Vladan Devedzic, "Using Ontologies in Design of Multimodal User Interfaces", in Matthias Rauterberg, Marino Menozzi, Janet Wesson (Eds.), Human-Computer Interaction - Proceedings of the IFIP INTERACT '03 Conference, IOS Press, pp. 535-542.
16. R. Prates, C. de Souza, S. Barbosa, "A Method for Evaluating the Communicability of User Interfaces", ACM Interactions January/February 2000, pp. 31-38.
17. Jovanov, D. Starčević, A. Samardžić, A. Marsh, Ž. Obrenović, "EEG analysis in a telemedical virtual world", Future Generation Computer Systems 15 (1999), pp. 255-263.
18. Zeljko Obrenovic, Dusan Starcevic, Bran Selic, "A Model Driven Approach to Content Repurposing", IEEE Multimedia, Vol. 11, No. 1, January-March 2004, pp. 62-71.

Zeljko Obrenovic received his PhD in computer science from the University of Belgrade. He holds a position of a researcher scientist at the Center for Mathematic and Computer Sciences in Amsterdam. His main research interests include human-computer interaction, semantic multimedia, ambient intelligence, biomedical engineering, and software methodologies. Contact address: Kruislaan 413, NL-1098 SJ Amsterdam, The Netherlands, Email: Zeljko.Obrenovic@cwi.nl.

Dusan Starcevic received his PhD in information systems from the University of Belgrade, Yugoslavia. He is currently a professor at the School of Business Administration, University of Belgrade, and a visiting professor at School of Electrical Engineering. His main research interests include distributed information systems, multimedia, and computer graphics. Contact address: FON - School of Business Administration, Jove Ilića 154, 11000 Belgrade, Serbia and Montenegro, Email: starcev@fon.bg.ac.yu.